Information and Computation 214 (2012) 47–58

Contents lists available at SciVerse ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

No double discount: Condition-based simultaneity yields limited gain

Yoram Moses [a,*], Michel Raynal [b]

[a] Department of Electrical Engineering, Technion, Haifa 32000, Israel
[b] Institut Universitaire de France & IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France

ARTICLE INFO

Article history:
Received 17 September 2010
Revised 5 February 2012
Available online 14 February 2012

Keywords:
Agreement problem
Condition-based agreement
Consensus
Common knowledge
Distributed algorithm
Early decision
Lower bound
Modularity
Process crash failure
Round-based computation model
Simultaneity
Synchronous message-passing system

ABSTRACT

We consider the consensus problem in synchronous message-passing distributed systems. A celebrated result states that every protocol that is guaranteed to tolerate up to t crash failures has a worst-case execution in which some process does not decide before the end of $t+1$ rounds. A variant of the problem in which the set of input vectors is restricted is called *condition-based consensus*. In this setting, Mostéfaoui, Rajsbaum and Raynal defined a natural degree of restriction called the *condition* of the set of input vectors that a protocol is assumed to handle. The condition is a natural number $d \leqslant t$, with a larger condition implying a smaller set of input values. Moreover, they showed that condition-d consensus can be solved in $t+1-d$ rounds in the worst case.

Dwork and Moses considered *simultaneous consensus*, a variant of (unconditional) consensus in which all correct processes must decide in the same round. Like ordinary consensus, this problem can be solved in $t+1$ rounds in the worst case. However, they showed that the stopping time depends on the pattern in which failures occur. They defined a notion of the *waste* $W(F)$ of a failure pattern F (where $0 \leqslant W(F) \leqslant t-1$), and showed that $t+1-W(F)$ rounds are necessary *and sufficient* for simultaneous consensus. They presented a solution that was optimal in *all* cases, and not just in the worst case: For every behavior of the adversary, their protocol stops as soon as *any* correct protocol can possibly stop.

This paper considers condition-based simultaneous consensus in the synchronous model.[1] It first presents a simple algorithm in which processes decide simultaneously at the end of the round $RS_{t,d,F} = (t+1) - \max\{W(F), d\}$. Then, the main result of the paper is presented, namely the statement and the proof that $RS_{t,d,F}$ is a lower bound for simultaneous condition-based consensus. This shows that, contrary to what could be hoped, when considering condition-based consensus with simultaneous decision, we can benefit from the best of both actual worlds (either the failure world when $RS_{t,d,F} = (t+1) - W(F)$, or the condition world when $RS_{t,d,F} = t+1-d$), but we cannot benefit from the sum of savings offered by both. Only the best discount applies. From a technical point of view, the lower bound result is based on two new notions associated with conditions on input vectors, called *d-coverability* and *d-tightness*.

1. Introduction

The consensus problem. Fault-tolerant systems often require a means by which processes or processors can arrive at an exact mutual agreement of some kind [14]. Indeed, reaching agreement is a fundamental primitive at the root of coordination

* Corresponding author.
E-mail addresses: moses@ee.technion.ac.il (Y. Moses), raynal@irisa.fr (M. Raynal).

[1] A preliminary version of this paper appeared at DISC'2008 (Moses and Raynal, 2008 [10]).

0890-5401/$ – see front matter
doi:10.1016/j.ic.2012.02.006

among processes in distributed systems. The agreement requirement is captured by the *consensus* problem that is one of the most important problems of fault-tolerant distributed computing. It actually occurs every time entities (usually called agents, processes—the word we use in the following—nodes, sensors, etc.) have to agree. The consensus problem is surprisingly simple to state: Each process is assumed to propose a value, and all the processes that are not faulty have to agree/decide (decision), on the same value (agreement), that must in turn be one of the proposed values (validity). The considered in this paper is the process crash model.

While consensus is impossible to solve in pure asynchronous systems prone even to a single process crash [4] ("pure asynchronous systems" means systems in which there is no upper bound on process speed and message transfer delay), it can be solved in synchronous systems (i.e., systems where there are such upper bounds) whatever the number n of processes and the number t of process crashes ($t < n$).

An important measure for a consensus algorithm is the time it takes for the non-faulty processes to decide. As a computation in a synchronous system can be abstracted as a *sequence of rounds*, the time complexity of a synchronous consensus algorithm is measured in terms of the minimal number of rounds (R_t) that a process participates in before deciding, in the worst case scenario. It has been shown (see, e.g., in [8]) that $R_t = t + 1$. Moreover, this bound is tight: There exist algorithms where no process ever executes more than R_t rounds (e.g., [15]); these algorithms are thus optimal with respect to the worst-case bound.

Early decision. While $t + 1$ rounds are needed in the worst case scenario, the major part of the executions have few failures or are even failure-free. So, an important issue is to be able to design *early deciding* algorithms, i.e., algorithms in which the processes decide "as early as possible" in good scenarios. Let f, $0 \leqslant f \leqslant t$, be the actual number of process crashes in an execution. It has been shown that the lower bound on the number of rounds is then $R_{t,f} = \min(f + 2, t + 1)$ (e.g., [1,8]). As before, this bound is tight: Algorithms in which no process ever executes more than $R_{t,f}$ rounds exist (e.g., see [1,15]).

Condition-based approach and the hierarchy of synchronous conditions. The condition-based approach was originally introduced to circumvent the impossibility of solving consensus in an asynchronous system prone to crash failures [12]. It consists in restricting the set of input vectors of values that can be proposed (such a set is called a *condition*). A main result associated with this approach is the following [12]: A condition C allows consensus to be solved in an asynchronous system prone to up to x process crashes iff it is x-legal. (Roughly speaking, a condition is x-legal if each of its input vectors contains a particular value more than x times, and the Hamming distance between two vectors from which different values can be decided is greater than x.) There is a strong connection between the condition-based approach and error-correcting codes [5].

While the condition-based approach is useful to extend computability of consensus in asynchronous systems, it has also been used to allow for faster agreement in synchronous systems [13]. More precisely, let us consider a synchronous system where up to t processes can crash, and let $\mathcal{C}^d$ be the set of all d-legal conditions, $\mathcal{C}^t \subset \cdots \subset \mathcal{C}^d \subset \cdots \subset \mathcal{C}^0$. For every condition $C \in \mathcal{C}^d$, it is shown in [13] that synchronous consensus can be solved uniformly for all input vectors $I \in C$ in one round when $d = t$, and in $t + 1 - d$ rounds when $0 \leqslant d \leqslant t$. It is also shown that $t + 1 - d$ is a tight worst-case lower bound.

Simultaneous decision. Consensus is a data agreement property, namely, processes must agree on the same value. Depending on the actual failure pattern, and the way this pattern is perceived by the processes, it is possible for several processes to decide in distinct rounds. The *simultaneous (decision)* consensus problem aims at eliminating this uncertainty. It requires that the processes decide on the same value (*data agreement*), during the very same round (*time agreement*).

Many "classical" consensus algorithms where all the processes that do not crash decide systematically at the end of the round $R_t = t + 1$ ensure simultaneous decision. Simultaneous consensus was considered in [2], where it was shown that simultaneous decisions can be attained in $RS_{t,F}$ rounds, $2 \leqslant RS_{t,F} \leqslant t + 1$, a number of rounds that depends on the failure pattern. Indeed, for every failure pattern F there is a measure called the *waste*, denoted by $W = W(F)$, such that decisions can be performed in $RS_{t,F} = t + 1 - W$ rounds, and no earlier. $W(F)$ represents the number of rounds "wasted" by the adversary in the failure pattern F. Interestingly, the greatest values of W are obtained when many processes crash early. If at most one process crashes in each round, $W = 0$ and simultaneous decision cannot be performed before the end of round $t + 1$. At first glance, this may appear counter-intuitive. Actually it is not; roughly speaking, it is a consequence of the fact that once many processes crash, the adversary options for the remaining rounds become more restricted. Simultaneous decision requires common knowledge about proposed input values, which becomes easier to attain as the uncertainty about past and future failures is reduced. Algorithms that solve simultaneous consensus optimally—requiring precisely $t + 1 - W(F)$ rounds for every run and each failure pattern F—are described in [2,7,9,11].

Content of the paper. This paper, which is an extended version of [10], investigates the simultaneous decision requirement in the context of the condition-based approach. Let $C \in \mathcal{C}^d$ (thus, C is d-legal). On the one hand, we have from [13] that simultaneous consensus can be guaranteed in round $t + 1 - d$. On the other hand, since a solution for unrestricted simultaneous consensus is still correct under any condition C, the existing solution to simultaneous consensus from [2,9,11] can be used to guarantee simultaneous agreement in $t + 1 - W$ rounds. This paper investigates the interaction between conditioning and simultaneity. For a positive result, it shows that the simultaneity requirement allows a seamless combination of the two solutions. Essentially, it is possible to simulate both types of algorithms and decide (without violating agreement!) in the round in which the first of the two algorithms decides. This solves condition-based simultaneous consensus

in $t+1-\max\{W,d\}$ rounds. On the negative side, it shows that, in a precise sense, this is the best possible. For natural d-legal conditions, the algorithm presented here is optimal in a very strong sense, inherited from the optimal solutions for simultaneous consensus: For each and every input vector I and failure pattern F, the algorithm decides as soon as any other simultaneous consensus algorithm for C would decide in a run with I and F. So the algorithm is optimal *in every case* and not just in the *worst* case run. The main technical challenge (and main contribution) of the paper is in the lower-bound proof (presented in Section 4), which relies on a knowledge-based analysis, using the connection between simultaneous agreement and common knowledge [2,3,6].

The notion of d-legality (introduced in [12]) is crucial for the condition-based approach. It expresses the fact that two input vectors from which different values are decided, must be far enough one from the other (from Hamming distance point of view). From a technical point of view, the paper introduces two new notions related to input vectors, namely *d-coverability* and *d-tightness*. d-Coverability can be seen as a proximity property on input vectors in a condition, that turns out to be a central notion to prove the lower bound result. d-Tightness corresponds to a condition being both d-legal and $(d+1)$-coverable. For d-tight conditions, our condition-based simultaneous consensus algorithm for d-legal conditions matches the lower bounds for $(d+1)$-coverable conditions, leading to tight bounds and a solution that yields the best possible behavior in each and every execution.

Roadmap. The paper is made up of five sections. Section 2 presents the synchronous message-passing model, and the condition-based simultaneous consensus problem. Section 3 presents an optimal condition-based simultaneous consensus algorithm. This algorithm, that is a simple combination of two existing algorithms, is presented to make the paper self-contained. The main result of the paper, namely the establishment of the lower bound, is presented in Section 4. Finally, Section 5 concludes the paper.

2. Computation model, conditions and problem specification

2.1. Computation model

Round-based synchronous system. The system model consists of a finite set of processes, denoted $\Pi=\{p_1,\ldots,p_n\}$, that communicate and synchronize by sending and receiving messages through channels. (We sometimes also use p and q to denote processes.) Every pair of processes is connected by a bi-directional reliable channel (which means that there is no creation, alteration, loss or duplication of messages).

The system is *round-based synchronous*. This means that each execution consists of a sequence of *rounds*. Those are identified by the successive integers $1, 2$, etc. For the processes, the current round number appears as a global variable r that they can read, and whose progress is managed by the underlying system. A round is made up of three consecutive phases:

- A send phase in which each process sends the same message to all the processes (including itself).
- A receive phase in which each process receives messages. The fundamental property of the synchronous model lies in the fact that a message sent by a process p_i to a process p_j in round r, is received by p_j in the very same round r.
- A computation phase during which each process processes the messages it received during that round and executes local computation.

Process failure model. A process is *faulty* during an execution if its behavior deviates from that prescribed by its algorithm, otherwise it is *correct*. We consider here the crash failure model, namely, a faulty process stops its execution prematurely. A process can fail at most once, and after it has crashed, the process does nothing. If a process crashes in the middle of a sending phase, only a subset of the messages it was supposed to send might actually be received.

As already indicated, the model parameter t $(0\leqslant t\leqslant n-1)$ denotes an upper bound on the number of processes that can crash in a run.[2] A *failure pattern* F is a set of triples $\langle q,k_q,B_q\rangle$. A triple $\langle q,k_q,B_q\rangle$ states that the process q crashes while executing the round k_q (hence, it sends no messages after round k_q), while the set B_q denotes the set of processes that do not receive the message sent by q during the round k_q. $\mathsf{Fails}(t)$ denotes the set of failure patterns with at most t triples. An upper bound of t on the number of failures will translate into the assumption that all runs will have failure patterns from $\mathsf{Fails}(t)$.

Inputs. Let $\mathcal{V}$ be the set of values that can be proposed, with $|\mathcal{V}|\geqslant 2$. An *input configuration* is an assignment $\mathcal{I}:\Pi\to\mathcal{V}$ of an initial value $v_i\in\mathcal{V}$ to each process p_i. An *input vector* is a size n vector corresponding to an input configuration. A *condition* is a set C of input vectors.

[2] We assume that $t<n-1$ rather than $t<n$ for ease of exposition. In the boundary case of $t=n-1$, simultaneous (unconditional) consensus can be obtained in precisely $n-1-W(F)=t-W(F)$ rounds, as opposed to $t+1-W(F)$ rounds for $t<n-1$. See Sections 6.6 and 6.8 of [3] for more details. The double discount results of this paper apply equally well in the case $t=n-1$, but stating them for both $t=n-1$ and $t<n-1$ would be somewhat cumbersome.

Algorithms. Before round 1 each process p_i is in an *initial state* consisting of its initial value v_i (the ith component of the input vector). A system *configuration* or global state before round r, is a sequence $(s_1, s_2, \ldots, s_n)$ consisting of the local states of the processes $1, 2, \ldots, n$ at that point. The input vector I is the configuration of the system before the first round, and a new configuration is obtained after every round as follows.

In this paper we focus on *deterministic* algorithms for consensus. For concreteness, we identify such an algorithm with a function that specifies, for each process, a function from the local state of a process after receiving its round r messages, that specifies its decisions in round r and the messages it sends in round $r+1$. Typically, our algorithms will have the form $P[n, t, C]$, taking as parameters the number of processes n, the bound t on the number of possible crashes, and when relevant also a set C of input vectors defining the condition under which the algorithm operates. A run of $P[n, t, C]$ will have an input vector $I \in C$. We write P for short, when the parameters are fixed or clear from context. In the synchronous model we are considering, an algorithm determines everything about the execution except for the initial global state of the processes and the failure pattern. $P(I, F)$ denotes the unique run of algorithm P when processes start out with initial values according to I (discussed further below), and when the failure pattern is F.

2.2. The condition-based approach

Notation. Let $\perp$ denote a default value such that $\perp \notin \mathcal{V}$ and $\forall a \in \mathcal{V}$, $\perp < a$. We usually denote by I an input vector (all its entries are in $\mathcal{V}$), and by J a vector that may have some entries equal to $\perp$. Such a vector J is called a *view*. The number of occurrences of a value $a \in \mathcal{V} \cup \{\perp\}$ in the vector J is denoted $\#_a(J)$. Given two vectors $I1$ and $I2$, $dist(I1, I2)$ denotes their Hamming distance.

Definition 1 *(d-Legal).* (See [5,12].) Let $d < t$ be a natural number. A condition C is *d-legal* if there exists a function $\mathcal{H} : C \mapsto \mathcal{V}$ satisfying: (1) $\forall I \in C$: $\#_{\mathcal{H}(I)}(I) > d$, and (2) $\forall I1, I2 \in C$: $\mathcal{H}(I1) \neq \mathcal{H}(I2) \Rightarrow dist(I1, I2) > d$.

This means that the value extracted from I by $\mathcal{H}()$ appears "often enough" in I (more than d times), and two vectors from which different values are extracted by $\mathcal{H}()$ are "far enough apart" in terms of Hamming distance. Intuitively, the function $\mathcal{H}()$ justifying a given condition selects a proposed value to become the decided value, namely, the value selected from an input vector I is $\mathcal{H}(I)$.

Example 1. Assuming that the set of initial values $\mathcal{V}$ is totally ordered, let $\max(I)$ denote the greatest value in an input vector I. The max-*value* condition, denoted M_d, is defined by:

$$M_d = \big\{I\colon \#_{\max(I)}(I) > d\big\}.$$

The max-value condition M_d is d-legal [12].

A straightforward consequence of the definition of d-legal conditions is that every d-legal condition is also $(d-1)$-legal. Moreover, it is easy to check that the condition M_d is not $(d+1)$-legal. Thus, the sets $\mathcal{C}^d$ of all d-legal conditions form a strict hierarchy, as captured by:

Theorem 1. *(See [12].) Let $0 \leqslant d \leqslant t-1$. The set $\mathcal{C}^{d+1}$ of $(d+1)$-legal conditions is strictly contained in the set $\mathcal{C}^d$ of (d)-legal conditions. Thus, $\mathcal{C}^t \subset \cdots \subset \mathcal{C}^d \subset \cdots \subset \mathcal{C}^0$.*

A main result of [12] is the characterization of the largest set of conditions that allow consensus to be solved in an asynchronous system: If C is d-legal then consensus can be solved under C in an asynchronous system prone to up to d process crashes. Conversely, there exists a $(d-1)$-legal condition C' for which consensus is unsolvable in the presence of d process crashes.

2.3. The condition-based simultaneous (cb-s) consensus problem

The problem we focus on in this paper has been informally stated in the introduction: Every process p_i *proposes* a value v_i (called its *initial value*). Assuming that the vector of proposed values belongs to a predetermined condition C, the processes have to *decide*, during the very same round, on the same value, which is required to be one of the proposed values. This can be stated as a set of four properties that any algorithm solving the problem has to satisfy.

- Decision. Every correct process decides.
- Validity. A decided value is a proposed value.
- Agreement. No two processes decide different values.
- Simultaneous decision. No two processes decide during distinct rounds.

3. An optimal condition-based simultaneous consensus algorithm

This section presents a simple condition-based simultaneous consensus algorithm in which the processes decide at the end of the round $RS_{t,d,F} = (t+1) - \max\{W, d\}$. Section 4 will show that $RS_{t,d,F}$ is the smallest number of rounds for condition-based simultaneous consensus.

The proposed algorithm is constructed in a modular fashion. It combines two base algorithms. One is a condition-based algorithm that, when instantiated with a d-legal condition C (i.e., $C \in \mathcal{C}^d$, with $0 \leqslant d \leqslant t$, and the input vector I belongs to C) directs the processes to decide simultaneously at the end of round $t+1-d$. The second is a simultaneous (non-condition-based) consensus algorithm in which the processes decide at the end of round $t+1-W$. These base algorithms are presented in Sections 3.1 and 3.2.

In general, obtaining a consensus algorithm by combining two consensus algorithms that execute concurrently is a subtle matter. When the two algorithms give rise to conflicting decisions, some rule for deciding among them is needed. This causes a genuine problem if different processes see the decisions in different orders. In our case, the fact that both algorithm guarantee simultaneous decisions simplifies this task. If one of the algorithms decides earlier, then its value can be adopted as that of the combined algorithm. Otherwise, both algorithms decide (everywhere) in the same round, and a fixed *a priori* rule can be used to determine which of the algorithms should take precedence. As a consequence, the combination of the two algorithms yields a cb-s consensus algorithm in which the processes simultaneously decide at the end of round $RS_{t,d,F} = (t+1) - \max\{W(F), d\}$.

3.1. A simple condition-based simultaneous consensus algorithm

3.1.1. Adapting an existing algorithm

It is convenient to consider partial input vectors, which specify some of the initial values, and contain $\perp$ instead of the missing values. We write $J \leqslant I$ if I is "more informative" than J, namely, (i) every location containing $\perp$ in I contains $\perp$ in J as well, and (ii) I and J agree on all locations that are not $\perp$ in J. An important property of d-legal conditions, which follows directly from the definition of legality and that will be useful for solving condition-based consensus is captured by the following lemma:

Lemma 1. *(See [13].) Let C be a d-legal condition and let $\mathcal{H}$ be the function justifying that it is d-legal. If $I1, I2 \in C$, $\#_\perp(J) \leqslant d$, $J \leqslant I1$ and $J \leqslant I2$, then $\mathcal{H}(I1) = \mathcal{H}(I2)$.*

Given Lemma 1, any subset of $n-d$ values in an input vector I determine the value of $\mathcal{H}(I)$. It is thus possible to extend $\mathcal{H}$ to partial input vectors that contain any number $k \geqslant n-d$ of values. Since fewer values suffice, it turns out that condition-based consensus admits faster algorithms in the synchronous round model than (unconditional) consensus. Indeed, Mostéfaoui et al. in [13] present a condition-based consensus algorithm for the synchronous round model that, when applied with respect to a condition in $\mathcal{C}^d$, guarantees that all non-faulty processes decide in the first $t+1-d$ rounds. Their algorithm, however, does not guarantee simultaneous decision.[3] The algorithm of [13] can be adapted so that decisions are performed simultaneously in round $t+1-d$, essentially by delaying all decisions until this round. The adapted algorithm COND_PROPOSE is presented in Fig. 1 and it satisfies these properties. By the analysis in [13], we have:

Theorem 2. *The* COND_PROPOSE *algorithm presented in Fig.* 1 *solves the condition-based simultaneous consensus problem. In every run in which the condition parameter for* COND_PROPOSE *is d, decision is reached in $(t+1)-d$ rounds.*

3.1.2. The COND_PROPOSE algorithm

Local variables. Each process p_i uses three local variables.

- V_i is an array whose aim is to contain p_i's local view of the input vector. Initialized to $[\perp, \ldots, \perp]$, it contains at most t entries equal to $\perp$ at the end of the first round (line 105).
- v_cond_i (initialized to $\perp$) is destined to contain the value $\mathcal{H}(I)$ that the condition C associates with the input vector I (line 106). As the condition C is d-legal, $\mathcal{H}(I)$ can be computed from $\mathcal{H}(V_i)$ only when the local view V_i of p_i has at most d entries equal to $\perp$. Thus, the value of $\mathcal{H}$ can be computed at this point.
- v_nocond_i (initialized to $\perp$) is destined to contain the value to be decided when no value can be decided from the condition because there are too many failures during the first round (more than d processes crash). When this occurs, a process will decide the greatest proposed value it has ever seen.

[3] The algorithm described in [13] works whether the input vector is in the condition or not. Let f be the actual number of failures that occur in a given run. When the input vector I belongs to the condition C, a process decides in two rounds if $f \leqslant d$, and in at most $t+1-d$ rounds when $f > d$. When the input vector is not in the condition a process decides in at most $t+1$ rounds. Differently, the condition-based algorithms developed here assume that the input vector is always from a condition of the appropriate class.

```
operation COND_PROPOSE(v_i, t, d, H):                    % code for p_i %
(101)  V_i ← [⊥, ..., ⊥]; v_cond_i ← ⊥; v_nocond_i ← ⊥;
(102)  when r = 1
(103)  begin round
(104)    send (v_i) to all;
(105)    for each v_j received do V_i[j] ← v_j end for;
(106)    if (#_⊥(V_i) ≤ d) then v_cond_i ← H(V_i) end if;
(107)    v_nocond_i ← max( all the v_j received during r)
(108)  end round;
(109)  when r = 2, 3, ... do
(110)  begin round
(111)    send (v_cond_i, v_nocond_i) to all;
(112)    v_cond_i   ← max( all the v_cond_j received during r);
(113)    v_nocond_i ← max( all the v_nocond_j received during r);
(114)    if (r = (t + 1) − d) then
(115)      if (v_cond_i ≠ ⊥) then return (v_cond_i) else return (v_nocond_i) end if
(116)    end if
(117)  end round.
```

Fig. 1. COND_PROPOSE: A synchronous condition-based simultaneous consensus algorithm.

Process behavior. The behavior of a process p_i is simple. During the first round (lines 102–108), p_i determines its local view V_i of the input vector I, computes v_cond_i if it does not see too many failures (i.e., at most d crashes), and computes v_nocond_i in case the condition is useless because there are more than d crashes. Then, from the second round until round $t+1-d$, the processes use the flood set strategy to exchange their current values v_cond_i and v_nocond_i, and keep the greatest ones of each. At the end of the round $t+1-d$, a process p_i decides on the value in v_cond_i if that value is not $\perp$; otherwise, it decides on the value in v_nocond_i (which is different from $\perp$).

3.2. An optimal algorithm for (unconditional) simultaneous consensus

The second basic algorithm that we shall use is taken from our paper [9], which is in turn based on [2,7]. It optimally solves the simultaneous consensus problem: For every failure pattern F and input vector I, it decides as soon as any other algorithm can decide on (F, I). Before describing the algorithm, we need a few definitions.

3.2.1. Preliminary definitions

For simplicity we assume that each process sends a message to all the processes in each round. As a consequence, process failures can be easily detected, and this detection is done as soon as possible. Moreover, for every algorithm P, every pair (I, F) consisting of and input vector and failure pattern determines a unique run ρ of P; we write $\rho = P(I, F)$ in this case.

Definition 2 *(Failure exposure).* With respect to a failure pattern F, the failure of a process q is *exposed* in round r if r is the first round in which there is a process p that (1) is blocked by F from receiving a round r message from q, and (2) p survives (i.e., completes without crashing) round r. Process q has *detectably crashed* in F by round r if its failure is exposed no later than in round r.

If q is exposed in round r according to the above definition, then in an algorithm in which q is always required to send a round r message to p (as in the case of full-information algorithms), process q's failure will be discovered by p: Process p knows at the end of round r that q has crashed.

Definition 3 *(Waste).* Fix a failure pattern F. Define by $\mathbf{C}(F, r)$ the number of processes whose failure is exposed in F by round r. Moreover, we use the convention that $\mathbf{C}(F, 0) = 0$. Let the *Waste inherent* in F, denoted $W(F)$, be defined as follows:

$$W(F) = \max_{r \geqslant 0} \big\{ \mathbf{C}(F, r) - r \big\}.$$

It immediately follows that $0 \leqslant W(F) \leqslant t-1$ holds for all failure patterns F. (We sometimes write W for $W(F)$ when there is no confusion.)

3.2.2. The SIM_PROPOSE algorithm

As in the condition-based case, we present for the sake of completeness the algorithm called SIM_PROPOSE, taken from [9], which will be our concrete optimal unconditional simultaneous consensus algorithm. (For more details and a proof of optimality, see [9].)

```
algorithm SIM_PROPOSE(v_i, t):                    % code for p_i %
(201)  est_i ← v_i; bh_i[0] ← t + 1; f_i[0] ← ∅; decided_i ← false;   % initialization %
(202)  when r = 1, 2, ... do                      % r: round number %
(203)  begin round
(204)      send (est_i, f_i[r − 1]) to all;       % including p_i itself %
(205)      let f'_i[r − 1] = the union of the f_j[r − 1] sets received during r;
(206)      let f_i[r] = the set of processes. from which p_i has not received a message during r;
(207)      est_i ← max( all the est_j received during r);
(208)      let h_i(r) = (r − 1) + (t + 1 − |f'_i[r − 1]|);
(209)      bh_i[r] ← min (bh_i[r − 1], h_i(r));
(210)      if r = bh_i[r] then decided_i ← true; return (est_i) end if
(211)  end round.
```

Fig. 2. SIM_PROPOSE: Optimal simultaneous consensus despite up to t crash failures.

The horizon notion. The algorithm is based on the notion of a *horizon* that each process computes in every round. Very roughly speaking, the horizon is a current best estimate for when initial values will become common knowledge, so that decisions can be based on them. The horizon is updated to be the minimal value of $t+1-|f'_i(r-1)|$, where the latter term $f'_i(r-1)$ refers to the set of processes that, according to p_i's knowledge at the end of round r were definitely exposed by the end of round $r-1$.

Local variables. Each process p_i manages the following local variables. Some variables are presented as belonging to an array. This is only for notational convenience, as such array variables can be implemented as simple variables.

- est_i contains, at the end of round r, process p_i's current estimate of the decision value. Its initial value is v_i, the value proposed by p_i.
- $decided_i$ is a write-once variable, initially undefined, that depicts process i's decision value in the current run.
- $f_i[r]$ denotes the set of processes from which p_i has not received a message during the round r. (So, this variable is the best current estimate that p_i can have of the processes that have crashed.)
  Let $\overline{f_i[r]} = \Pi \setminus f_i[r]$ (i.e., the set of processes from which p_i has received a round r message).
- $f'_i[r-1]$ is a value computed by p_i during the round r, but that refers to crashes that occurred up to the round $r-1$ (included), hence the notation. It is the value $\bigcup_{p_j \in \overline{f_i[r]}} f_j[r-1]$, which means that $f'_i[r-1]$ is the set of processes that were known as crashed at the end of the round $r-1$ by at least one of the processes from which p_i has received a round r message. As a process p_i receives its own messages, we have $f_i[r-1] \subseteq f'_i[r-1]$.
- $h_i(r)$ is the horizon value computed by i for round r.
- $bh_i[r]$ represents the best (smallest) horizon value known by p_i at round r. It is p_i's best estimate of the smallest round for a simultaneous decision. Initially, $bh_i[0] = h_i(0) = t+1$.

Process behavior. Each process p_i not crashed at the beginning of r sends to all the processes a message containing its current estimate of the decision value (est_i), and the set $f_i[r-1]$ of processes it currently knows as faulty. After it has received the round r messages, p_i computes the new value of est_i and the value of $bh_i[r]$. The new value of est_i is the smallest of the estimates values it has seen so far. As far as the value of $bh_i[r]$ is concerned, we have the following.

- The computation of $bh_i[r]$ must take $h_i(r)$ into account. This is required to benefit from the fact that there is a clean round y such that $r \leqslant y \leqslant h_i(r)$. In a *clean* round, all processes hear from the same set of processes. Following such a clean round, any two (non-crashed) processes p_i and p_j will have $est_i = est_j$, and (as they will receive messages from the same set of processes) will also satisfy that $f'_i[r-1] = f'_j[r-1]$. Consequently, we will have $h_i(y) = h_j(y)$, thereby creating correct "seeds" for determining the smallest round for a simultaneous decision. This allows the processes to determine rounds at which they can simultaneously decide.
- As we are looking for the first round where a simultaneous decision is possible, $bh_i[r]$ has to be set to $\min(h_i(0), h_i(1), \ldots, h_i(r))$, i.e., $bh_i[r] = \min(bh_i[r-1], h_i(r))$.

Finally, according to the previous discussion, the algorithm directs a process p_i to decide at the end of the first round r that is equal to the best horizon currently known by p_i, i.e., when $r = bh_i[r]$.

The resulting algorithm is presented in Fig. 2, where $h_i(r)$ (see line 208) is expressed as a function of $r-1$ to emphasize the fact that it could be computed at the end of the round $r-1$ by an external omniscient observer. We thus obtain:

Theorem 3. *(See [2].) Let $t < n-1$. The* SIM_PROPOSE *algorithm of Fig. 2 solves simultaneous consensus. In every run, decision is reached in round $RS_{t,F} = t+1-W$. Moreover, no simultaneous consensus algorithm can ever decide in fewer than $t+1-W(F)$ in a run with failure pattern F.*

3.3. An optimal condition-based simultaneous consensus algorithm

This section shows how the two simultaneous consensus algorithms presented in Figs. 1 and 2 can be combined to give a correct simultaneous consensus algorithm that decides as soon as either one does. In fact, a similar combination can be done in general for simultaneous consensus algorithms. (The issue of parallel execution of consensus algorithms without the simultaneity property is much more subtle and is not always possible.)

Running simultaneous consensus algorithms in parallel. Suppose that A and B are two algorithms for simultaneous consensus, whose decision round is determined by conditions ψ_A and ψ_B, respectively. Define A else B to be the algorithm that runs both A and B in parallel, but changes the decision rule in the following way:

- When algorithm A's decision rule ψ_A is satisfied, then decision is performed according to A, provided that no decision has been performed in previous rounds.
- If (i) no decision has been performed in previous rounds, (ii) A's decision rule ψ_A is *not* satisfied, and (iii) B's decision rule ψ_B *is* satisfied, then decision is performed according to B.

Thus, A else B decides as soon as the first of A and B decide, and when both algorithms happen to decide in the same round, A's decision is adopted. Moreover, all properties of simultaneous consensus (decision, validity, agreement and simultaneity) are satisfied by A else B.

The optimal algorithm. The changes to the algorithms in Figs. 1 and 2 that yield a properly combined algorithm are the following ones.

1. The r-th round, $1 \leqslant r \leqslant t+1-d$, of the combined algorithm is a simple merge of the r-th round of both algorithms. This means that the message sent by p_i at round r now piggybacks v_cond_i, v_nocond_i, est_i and $f_i[r-1]$ (a closer look at the base algorithms shows that their variables v_nocond_i and est_i play the same role, and consequently, only one of them needs to be kept in the combined algorithm).
2. Lines 114–116 of the algorithm in Fig. 1, and line 210 of the algorithm in Fig. 2 are replaced by the following lines:

```
if (r = bh_i[r]) ∨ (r = t + 1 − d) then
  if (r = bh_i[r]) then decided_i ← true; return (est_i)
                   else if (v_cond_i ≠ ⊥) then return (v_cond_i)
                                          else return (v_nocond_i) end if
end if.
```

We call the resulting combined algorithm Cond_Sim_Propose, and immediately obtain:

Theorem 4. *Let $t < n-1$. The* Cond_Sim_Propose *algorithm solves the condition-based simultaneous consensus problem. Moreover, it decides in round $t+1-\max\{W(F), d\}$ in a run with failure pattern F.*

4. Optimality: $t+1-\max\{W, d\}$ is a lower bound

This section is the heart of the paper. It proves that the Cond_Sim_Propose algorithm described in Section 3.3 is optimal. Namely, in a synchronous system prone to up to t process crashes (with $t < n-1$), there is no deterministic algorithm that can ever solve the condition-based simultaneous consensus problem in fewer than $(t+1)-\max\{W, d\}$ rounds. However, a precise formulation of this claim is somewhat subtle. To see why, suppose that C is a d-legal condition. We know that Cond_Sim_Propose, running with condition parameter d, will halt in $t+1-\max\{W, d\}$ rounds in runs with inputs from C. Recall, however, that every $(d+1)$-legal condition is, in particular, also d-legal. Hence, C could also be $d+1$ legal, in which case both Cond_Propose and Cond_Sim_Propose, running with parameter $d+1$, will halt in $t-d$ in some runs with waste $W=0$, improving on $(t+1)-\max\{W, d\}$ since $t-d < t+1-d = t+1-\max\{d, 0\}$. In fact, d-legality is a property that requires input vectors to be sufficiently far from one another. A lower bound will be based on a property called *k-coverability* that says something about how close vectors are. We will show that no algorithm can halt in fewer than $t+1-\max\{W, d\}$ rounds if the condition C is $(d+1)$-coverable. Finally, we will define the class of *d-tight* conditions, which are both d-legal and $(d+1)$-coverable, for which Cond_Sim_Propose is optimal in all runs: In every run, it decides as soon as it is allowed by the lower bound *for the particular failure pattern* in the run.

4.1. Similarity graphs and common knowledge

Our lower bound is based on the well-known connection between simultaneous agreement and common knowledge [2,11]. This connection implies that it is possible to simultaneously decide on a value $v \in \mathcal{V}$ only once it becomes common knowledge that one of the initial values of the current input vector is v. Roughly speaking, if no value is commonly known

to be an initial value in round r of a run ρ, then decision is not possible in this round. We will use this in order to prove lower bounds on when decisions can be performed in a cb-s consensus algorithm. Rather than develop the logic of knowledge in detail here, we will employ a simple graph-theoretic interpretation of common knowledge that applies in our setting.

The *local state* of process p_i at the end of round r of an algorithm P is identified with the contents of p_i's memory—its variables and their values. Intuitively, we think of runs ρ and ρ' as being *indistinguishable to q after round r* if q's local state at the end of round r is the same in both runs. We use this notion of *indistinguishability* to define a similarity graph over runs of an algorithm, adjusted for the condition-based setting.

Definition 4 *(Condition-based similarity graph).* Fix an algorithm $P = P[n, t, C]$. We define the similarity graph for P after round $r \geqslant 0$, denoted by $\mathsf{csG}(r)$, to be the undirected graph (V, E), where

(i) V consists of the set of runs $\rho = P(I, F)$ with $I \in C$ and $F \in \mathsf{Fails}(t)$, while
(ii) $\{\rho, \rho'\} \in E$ iff there is some process q that survives round r in both ρ and ρ' and has the same local state at the end of round r in both runs.

We say that runs ρ and ρ' are *connected after round r*, and write $\rho \overset{r}{\approx} \rho'$, if ρ and ρ' are in the same connected component of $\mathsf{csG}(r)$. Notice that the $\overset{r}{\approx}$ relation is reflexive, since connected components define an equivalence relation. Connectivity in the similarity graph is closely related to common knowledge [2]. In fact, one way to formally define common knowledge in our setting is via the similarity graph: The fact A about the run is *common knowledge* at the end of round r of a run ρ if A holds at all runs $\rho' \overset{r}{\approx} \rho$ (i.e., if A holds at all runs that are r-connected to ρ). An equivalent formulation of the fact that a value v must be common knowledge before it can be decided on in a simultaneous consensus protocol is thus expressed as follows:

Lemma 2. *(See [2].) Assume that $P = P[n, t, C]$ solves cb-s consensus, and let $\rho = P(I, F)$, where $I \in C$ and $F \in \mathsf{Fails}(t)$. Moreover, assume that the correct processes decide on value v in round r of ρ. Then v appears in the input vector of every run $\rho' \overset{r}{\approx} \rho$.*

From Lemma 2, we immediately obtain:

Corollary 1. *Fix r, let $I_{\bar{v}} \in C$ be an input vector that does not contain v as an initial value. Moreover, let $\rho = P(I, F)$ and $\rho' = P(I_{\bar{v}}, F')$ be runs of a cb-s consensus algorithm P. If $\rho \overset{r}{\approx} \rho'$ then no process can decide v in round r of ρ.*

The analysis in [2] showed that unconditional simultaneous consensus is possible at the end of round $t + 1 - W(F)$ in runs with failure pattern F, but it cannot be reached at an earlier round. Thus, following [9], we call a round r *premature* in a failure pattern F if $r < t + 1 - W(F)$. It was shown in [2] that before round $t + 1 - W(F)$, then, roughly speaking, the only fact about failures that is common knowledge in a run with failure pattern F is that $r < t + 1 - W$, where W refers to "waste in the current run's failure pattern". It is not common knowledge that even one failure has occurred. More formally, in our setting this becomes:

Lemma 3. *(See [2,9].) Fix an algorithm $P = P[n, t, C]$, and let $\rho = P(I, F)$ and $\rho' = P(I, F')$, where $I \in C$ and $F, F' \in \mathsf{Fails}(t)$. If round r is premature in both F and F', then $\rho \overset{r}{\approx} \rho'$.*

The corresponding lemma in [9] is not stated in terms of a condition C, but its proof depends only on the fact that the similarity graph contains all runs $\hat{\rho} = P(I, \hat{F})$ with failure patterns $\hat{F} \in \mathsf{Fails}(t)$. Since this is true for $\mathsf{csG}(r)$ as well, the same proof, unchanged, establishes Lemma 3 as stated here.

The combination of Lemma 2 and Lemma 3 suggests that studying the class of runs in which r is premature can provide insight into the solvability of cb-s consensus. For example, let us consider the question of how many processes can be silenced from the outset (initially crashed) for round r to be premature in a failure pattern $\hat{F}$. Formally, a process q is crashed from the outset if the failure pattern contains the triple $(q, 1, \Pi)$. Clearly, if k processes are crashed in $\hat{F}$ from the outset and no other failures occur, then $W(\hat{F}) = k - 1$. By definition, round r is premature if $r < t + 1 - W(\hat{F})$, which in our case translates into $r < t + 1 - (k - 1) = t + 2 - k$ or, equivalently, if $k < t + 2 - r$. We can use this fact to derive the following consequence of Lemma 3:

Lemma 4. *Fix an algorithm $P = P[n, t, C]$ and let $\rho = P(I, F)$ and $\rho' = P(I', F')$. Moreover, let $dist(I, I') \leqslant k$ and suppose that round r is premature in both F and F'. If $r < t + 2 - k$ then $\rho \overset{r}{\approx} \rho'$.*

Proof. Assume that the conditions of the lemma hold, and let $r < t + 2 - k$. Thus, in particular, I and I' differ at most on the set of values assigned to processes $p_{i_1}, \ldots, p_{i_k}$. Consider the failure pattern $\hat{F}$ in which $p_{i_1}, \ldots, p_{i_k}$ are crashed and

silent from the outset, while no other processes fail in $\hat{F}$. Recall that $W(\hat{F}) = k-1$. Since, by assumption, we have that $r < t+2-k$, it follows that $r < t+1-(k-1) = t+1-W(\hat{F})$, and so r is premature in $\hat{F}$. We define two runs $\rho_1 = P(I, \hat{F})$ and $\rho_2 = P(I', \hat{F})$ with the same input vectors as ρ and ρ', respectively, and with failure pattern $\hat{F}$. By Lemma 3 we have that $\rho \overset{r}{\approx} \rho_1$ as well as that $\rho' \overset{r}{\approx} \rho_2$, and by symmetry of $\overset{r}{\approx}$ the latter implies that $\rho_2 \overset{r}{\approx} \rho'$. Given that $t < n-1$ there is at least one process q that is non-faulty in $\hat{F}$, and thus also in both ρ_1 and ρ_2. Observe that all non-crashed processes have the same local state at the end of all rounds (and in particular process q at the end of round r) of both runs. We thus immediately have that $\{\rho_1, \rho_2\}$ is an edge of $\mathsf{csG}(r)$, and so, in particular, $\rho_1 \overset{r}{\approx} \rho_2$. Combining the above claims we obtain that $\rho \overset{r}{\approx} \rho_1 \overset{r}{\approx} \rho_2 \overset{r}{\approx} \rho'$, which by transitivity of '$\overset{r}{\approx}$' yields that $\rho \overset{r}{\approx} \rho'$, as desired. □

Lemma 4 can be used to derive additional properties about r-connectedness of runs. Suppose that in addition to ρ and ρ' as in Lemma 4 we have $\rho'' = P(I'', F'')$ where $dist(I', I'') \leqslant k$ and r is premature in F'' as well. Then, we can apply Lemma 4 to obtain that $\rho' \overset{r}{\approx} \rho''$ and by transitivity obtain that $\rho \overset{r}{\approx} \rho''$ as well. We can continue this process, and if it leads us to derive $\rho \overset{r}{\approx} \hat{\rho}$ where, in the latter, v is not an initial value, then v cannot be decided on in round r of ρ, by Corollary 1. We now formalize this intuition and use it to state and prove precise bounds for cb-s consensus.

Definition 5 *(Input k-graph of C).* Fix a condition C and a natural number k. The (*input*) *k-graph over C* is $\mathcal{G}_k[C] = (C, E_k)$, where $E_k = \{\{I, I'\}\colon\ dist(I, I') \leqslant k\}$.

Thus, two vectors of C are neighbors in $\mathcal{G}_k[C]$ exactly if they disagree on the values of at most k processes. We can now formalize the preceding discussion using reachability among input vectors in the k-graph of a condition C:

Theorem 5. *Fix a cb-s consensus algorithm $P = P[n, t, C]$, let $\rho = P(I, F)$ where $I \in C$ and $F \in \mathit{Fails}(t)$, and suppose that round r is premature in F. Moreover, assume that there is a path in $\mathcal{G}_k[C]$ connecting I to an input vector $I_{\bar{v}}$ that does not contain v as an initial value. If $k < t+2-r$ then no process can decide v in round r of ρ.*

Proof. Assume that the assumptions of the theorem hold, and denote $\rho' = P(I_{\bar{v}}, F)$ be the run with input vector $I_{\bar{v}}$ and the same failure pattern as ρ. By Corollary 1 it suffices to show that $\rho \overset{r}{\approx} \rho'$. We prove this by induction on the length h of the shortest path from I to $I_{\bar{v}}$ in $\mathcal{G}_k[C]$. The base case of $h = 0$ is trivial, since $\rho = \rho'$ and $\overset{r}{\approx}$ is, by definition, reflexive. For the inductive step, assume that $h > 0$ and that the claim holds for $h-1$. Thus, for some $I1 \in C$ we have that $dist(I, I1) \leqslant k$ and $I1$ has distance $h-1$ from $I_{\bar{v}}$ in $\mathcal{G}_k[C]$. Denote $\rho_1 = P(I1, F)$. Since ρ_1 has failure pattern F and we are given that r is premature in F, round r is premature in ρ_1. Hence, by the inductive assumption for $h-1$ we have that $\rho_1 \overset{r}{\approx} \rho'$. Moreover, since $dist(I, I1) \leqslant k$ we have by Lemma 4 that $\rho \overset{r}{\approx} \rho_1$. By transitivity of '$\overset{r}{\approx}$' we thus obtain that $\rho \overset{r}{\approx} \rho'$, as claimed. □

Theorem 5 will be used hereafter to show that if the initial vectors in a condition C are sufficiently close to each other, then decision cannot be obtained before round $t+1-\max\{W, d\}$. Recall that d-legality is essentially a property that ensures that the inputs in a condition are sufficiently *far* apart. We now revisit conditions and define the closeness property needed for our lower bound, and a further property for which we can obtain tight bounds for cb-s consensus.

4.2. Properties of conditions, revisited

Consider $M = \{\{v\}^n\colon\ v \in \mathcal{V}\}$ (the condition made up of the vectors in which all initial values are the same). Clearly, consensus can be solved (simultaneously) for M with no rounds of communication: every process can decide on its own initial value. Observe that M is d-legal for every $0 \leqslant d \leqslant t < n-1$. It follows that we have no hope of basing a nontrivial lower-bound on the notion of d-legality. The property of being d-legal is sufficient for the upper bounds stated in Theorem 2 (it imposes a requirement that input vectors are sufficiently distant from each other). We now use the k-graph to define a property on conditions, called *k-coverability,* which goes in the opposite direction and stipulates that the input vectors in a condition be sufficiently Hamming close.

Definition 6 *(k-Coverability).* A condition C is *k-coverable* if for every value $v \in \mathcal{V}$ and input vector $I \in C$ there is an input vector $I_{\bar{v}} \in C$ such that

(i) v does not appear in $I_{\bar{v}}$, and
(ii) I and $I_{\bar{v}}$ are in the same connected component of $\mathcal{G}_k[C]$.

Intuitively, k-legality is inconsistent with k-coverability. More formally, we can show:

Lemma 5. *A nonempty k-legal condition cannot be k-coverable.*

Proof. Let $k \geqslant 0$ and suppose by way of contradiction that $C \neq \emptyset$ is both k-legal and k-coverable. Moreover, let $\mathcal{H}$ be the choice function on C guaranteed by the definition of k-legal (Definition 1). Let $I \in C$, and let v be the value for which $\mathcal{H}(I) = v$. Since C is k-coverable, we have that there is some vector $I_{\bar{v}} \in C$ not containing v that is connected to I in the k-graph $\mathcal{G}_k[C]$. Clearly, $\mathcal{H}(I_{\bar{v}}) \neq v$ since Definition 1 states that if $I_{\bar{v}} = w$ then $\#_w(I_{\bar{v}}) > k$ and $\#_v(I_{\bar{v}}) = 0 \leqslant k$ by choice of $I_{\bar{v}}$. Since I and $I_{\bar{v}}$ are connected, there must be two neighboring vectors $I1, I2$ along the path connecting them such that $\mathcal{H}(I1) = v$ and $\mathcal{H}(I2) \neq v$. Since these are neighboring vectors we have by definition of $\mathcal{G}_k[C]$ that $dist(I1, I2) \leqslant k$. Conversely, the fact that $\mathcal{H}(I1) \neq \mathcal{H}(I2)$ implies by definition of legality that $dist(I1, I2) > k$; contradiction. □

Since k-coverability is designed to ensure sufficient proximity of input vectors in a condition, we can use this notion to prove a lower bound on the time at which cb-s consensus can be reached.

Lemma 6. *Fix a cb-s consensus algorithm $P = P[n, t, C]$ where C is $(d+1)$-coverable. Moreover, let $\rho = P(I, F)$ where with $I \in C$ and $F \in \mathit{Fails}(t)$, and let $r < t + 1 - \max\{W(F), d\}$. Then no value v can be decided on at the end of round r in ρ.*

Proof. Let P, C, d, ρ, I, C, t and r satisfy the assumptions of the lemma. Choose an arbitrary value $v \in \mathcal{V}$. Since C is $(d+1)$-coverable we have by definition that ρ's input vector I is connected in $\mathcal{G}_{d+1}[C]$ to an input vector $I_{\bar{v}}$ in which v does not appear. Thus, there is a finite path $I = I^1, I^2, \ldots, I^k = I_{\bar{v}}$ in $\mathcal{G}_{d+1}[C]$ connecting I with $I_{\bar{v}}$. Define $\rho^j = P(I^j, F)$ for $j = 1, \ldots, k$. We prove by induction on $j \leqslant k$ that ρ and ρ^j are in the same connected component of $\mathsf{csG}(r)$. The case $j = 1$ is trivial since by definition $\rho^1 = \rho$. For the inductive step, let $1 \leqslant j < k$ and assume that $\rho \overset{r}{\approx} \rho^j$. By assumption, $r < t + 1 - W(F)$, and so round r is premature in F. Hence, r is premature in both ρ^j and ρ^{j+1}. Since $r < t + 1 - d$ as well, we have that $r < t + 2 - (d+1)$. By construction, $dist(I^j, I^{j+1}) \leqslant d + 1$. The runs ρ^j and ρ^{j+1} thus satisfy the conditions of Lemma 4, and the lemma implies that $\rho^j \overset{r}{\approx} \rho^{j+1}$. Transitivity of '$\overset{r}{\approx}$' yields that $\rho \overset{r}{\approx} \rho^{j+1}$, completing the inductive step. The inductive proof yields, in particular, that $\rho \overset{r}{\approx} \rho^k$. Since $\rho^k = P(I^k, F)$ does not contain v as an initial value, it follows by Lemma 2 that v cannot be decided on at the end of round r of ρ. Since $v \in \mathcal{V}$ was chosen arbitrarily, we obtain that no value can be decided at the end of round r, which completes the proof of the lemma. □

While we have by Lemma 5 that a nontrivial k-legal condition cannot be k-coverable, it is possible for such a condition to be $(k+1)$-coverable. Indeed, we define:

Definition 7 *(k-Tightness).* A condition C is *k-tight* if it is both k-legal and $(k+1)$-coverable.

As mentioned in Example 1, the max-value condition $M_d = \{I\colon \#_{\max(I)}(I) > d\}$ is d-legal [12]. In fact, we can extend this result to show:

Lemma 7. *Let $0 \leqslant d \leqslant t$ and $|\mathcal{V}| > 1$. Then the* max-value *condition M_d for $\mathcal{V}$ is d-tight.*

Proof. Since M_d is d-legal (see Example 1), it remains to show that it is $(d+1)$-coverable. Let $I \in M_d$ be an input and let $v \in \mathcal{V}$. Since $|\mathcal{V}| > 1$, there exist $w \in \mathcal{V}$ with $w \neq v$. Denote $\{w\}^n$ by $I_{\bar{v}}$. We will show that $I_{\bar{v}}$ is in I's connected component of $\mathcal{G}_{d+1}$. Since $d + 1 \geqslant 1$, we can clearly change values of I to w one by one and remain in the connected component, as long as there are more than d values of $\max(I)$ in the input configuration. We consider two cases. If $w = \max(I)$, then this process immediately yields that I and $\{w\}^n = I_{\bar{v}}$ are in the same connected component, as desired. If $w \neq \max(I)$, then we can show that there is a path in $\mathcal{G}_{d+1}[M_d]$ from I to an input $J \in M_d$ consisting of $d+1$ values of $\max(I)$ and the remaining $n - (d+1)$ values are w. Thus, $dist(J, \{w\}^n) = d + 1$. It follows that J and $\{w\}^n = I_{\bar{v}}$ are adjacent nodes of $\mathcal{G}_{d+1}[M_d]$, and so I and $I_{\bar{v}}$ are in the same connected component. We conclude that M_d is $(d+1)$-coverable, as claimed. □

Thus, every class in the $\mathcal{C}^d$ hierarchy contains a d-tight condition. Indeed, the d-tight conditions form a natural subclass of the d-legal conditions that are not $(d+1)$-legal. From Lemma 6 and the properties of the Cond_Sim_Propose algorithm, we can now immediately conclude that Cond_Sim_Propose is optimal in all runs:

Theorem 6. *If C is a d-tight condition, then the* Cond_Sim_Propose *algorithm is optimal for cb-s consensus in every run. I. e., for every cb-s consensus algorithm P for C, every $I \in C$ and every failure pattern $F \in \mathit{Fails}(t)$, if $\rho = P(I, F)$ decides in round r, then* Cond_Sim_Propose *decides on (I, F) in a round $r' \leqslant r$.*

5. Conclusion

This paper focused on simultaneous decision in the condition-based consensus setting. It has presented two results. The first is a condition-based consensus algorithm in which processes decide simultaneously at the end of the round $RS_{t,d,F} = (t+1) - \max\{W(F), d\}$ where $W(F) \geqslant 0$ is a value that depends on the actual failure pattern F, and d depends on the

position of the condition C (the algorithm is instantiated with) in the hierarchy of legal conditions. While legality is a condition ensuring that input vectors in the condition are far from each other, we defined a complementary property on conditions called k-coverability. Our second result, that is the main contribution of the paper, is a proof that $RS_{t,d,F}$ is a lower bound on the number of rounds for the simultaneous condition-based consensus problem for a $(d+1)$-coverable condition. A condition is d-tight if it is both d-legal and $(d+1)$-coverable. Combining our upper and lower bounds shows that for the class of d-tight conditions, $RS_{t,d,F}$ is the best we could hope for: We can benefit from the best world provided by the actual run (failure world when $RS_{t,d,F} = (t+1) - W$ or condition world when $RS_{t,d,F} = (t+1) - d$), but the two benefits do not compose. There is no double discount.

Acknowledgments

The authors want to thank the referee for a thorough reading with insightful and constructive comments that helped us improve the presentation of the paper. The work of the first author was supported in part by Israel Science Foundation (ISF) under Grants 1339/05 and 1520/11. The work of the second author was partially supported by the French ANR project DISPLEXITY devoted to computability and complexity in distributed computing.

References

[1] D. Dolev, R. Reischuk, R. Strong, Early stopping in Byzantine agreement, Journal of the ACM 37 (4) (April 1990) 720–741.
[2] C. Dwork, Y. Moses, Knowledge and common knowledge in a Byzantine environment: Crash failures, Information and Computation 88 (2) (1990) 156–186.
[3] R. Fagin, J.Y. Halpern, Y. Moses, M.Y. Vardi, Reasoning about Knowledge, MIT Press, Cambridge, MA, 2003.
[4] M.J. Fischer, N. Lynch, M.S. Paterson, Impossibility of distributed consensus with one faulty process, Journal of the ACM 32 (2) (1985) 374–382.
[5] R. Friedman, A. Mostéfaoui, S. Rajsbaum, M. Raynal, Asynchronous agreement and its relation with error-correcting codes, IEEE Transactions on Computers 56 (7) (2007) 865–876.
[6] J.Y. Halpern, Y. Moses, Knowledge and common knowledge in a distributed environment, Journal of the ACM 37 (3) (1990) 549–587.
[7] T. Mizrahi, Y. Moses, Continuous consensus via common knowledge, Distributed Computing 20 (5) (2008) 305–321.
[8] Y. Moses, S. Rajsbaum, A layered analysis of consensus, SIAM Journal of Computing 31 (4) (2002) 989–1021.
[9] Y. Moses, M. Raynal, Revisiting simultaneous consensus with crash failures, Journal of Parallel and Distributed Computing 69 (4) (2009) 400–409.
[10] Y. Moses, M. Raynal, No double discount condition-based simultaneity yields limited gain, in: 22th Int'l Symposium on Distributed Computing (DISC'08), in: LNCS, vol. 5218, Springer-Verlag, 2008, pp. 423–437.
[11] Y. Moses, M.R. Tuttle, Programming simultaneous actions using common knowledge, Algorithmica 3 (1988) 121–169.
[12] A. Mostéfaoui, S. Rajsbaum, M. Raynal, Conditions on input vectors for consensus solvability in asynchronous distributed systems, Journal of the ACM 50 (6) (2003) 922–954.
[13] A. Mostéfaoui, S. Rajsbaum, M. Raynal, Synchronous condition-based consensus, Distributed Computing 18 (5) (2006) 325–343.
[14] L. Pease, R. Shostak, L. Lamport, Reaching agreement in presence of faults, Journal of the ACM 27 (2) (1980) 228–234.
[15] M. Raynal, Fault-Tolerant Agreement in Synchronous Message-Passing Systems, Morgan & Claypool Publ., ISBN 9781608455256, 2010, 170 pp.